

PATENT
Atty. Dkt. No. ROC920010085US1
MPS Ref. No.: IBMK10085

REMARKS

This is intended as a full and complete response to the Final Office Action dated February 22, 2005, having a shortened statutory period for response set to expire on May 22, 2005. Applicants submit this response to place the application in condition for allowance or in better form for appeal. Please reconsider the claims pending in the application for reasons discussed below.

Claims 1-26 are pending in the application. Claims 1-26 remain pending following entry of this response. Claims 1, 4, 11, 12, 16, 18, 19, and 26 have been amended.

The Examiner has requested that Applicants clarify a submission included in a prior response. In the prior response, Applicants inadvertently included the submission: "Applicants submit that the amendments and new claims do not introduce new matter" (Applicants' remarks, page 7, paragraph 3). The presentation of claims in the Response was correct; no new claims were added, and claims 1-26 remained pending. Thus, the statement at page 7, paragraph 3, was incorrect. Applicants apologize for any confusion that this may have caused the Examiner. As the Examiner proceeded to respond to the claims as presented in the response, however, Applicants believe that no further action is necessary.

Claim Rejections - 35 U.S.C. § 103

Claims 1-26 are rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Pat. No. 6,263,489 to *Olsen et al.* (hereinafter, "*Olsen*") in view of U.S. Pat. No. 6,161,216 to *Shagam*. Applicants traverse this rejection.

The Examiner has maintained the rejection of claims 1-26 under *Olsen*, in View of *Shagam*. Applicants respectfully disagree with the Examiner that *Olsen*, in view of *Shagam* discloses Applicants' invention.

Regarding claims 1, 12, and 16: generally, *Olsen* discloses methods for debugging machine code that has "been subjected to an optimizing action, wherein the machine code may have been reordered, duplicated, eliminated or transformed, so as

PATENT
Atty. Dkt. No. ROC920010085US1
MPS Ref. No.: IBMK10086

not to correspond with the programs source code order." *Olsen*, Abstract. The debugging techniques taught by *Olsen* allow the execution of machine code in the same sequence as the source code instructions, while the program is being debugged. *Olsen* 4:12-32. When the machine code instructions are reordered through optimization, two situations occur that can make it difficult to debug a computer program. *Olsen* discloses a debugging technique for managing these situations. A first situation occurs when machine code statements that occur before a breakpoint correspond to source code statements that appear after the breakpoint. In this situation, these machine code statements are not executed prior to reaching the breakpoint. *Olsen*, 4:12-44. A second situation occurs when machine code sequences appear after the breakpoint, but appear before the breakpoint in the source code. *Olsen*, 4:12-44. In this situation, machine code statements appearing after the breakpoint may be emulated to preserve a source-order execution of the program. Thus, the "debugging method of [*Olsen*]" hides the effects of machine code optimizations from the user during interactive debugging and enables recovery of expected values of variables at breakpoints." *Olsen* 3:41-44.

In contrast, Applicants claim for techniques for selecting points within a program being debugged to insert a break point. The Examiner asserts that:

Olsen expressly discloses a method for debugging a program (see, for example, the abstract) that includes the step of setting or inserting breakpoints in the program (see, for example, step 50 in FIG. 4a). When the user sets or inserts a breakpoint, the debugger likewise sets or inserts a corresponding breakpoint in the machine code (see, for example, column 5, lines 29-31)."

Final Office Action Page 2, Section 4. The material cited by the Examiner is one of several examples described in *Olsen* wherein a user sets a breakpoint, including:

"The user sets a breakpoint at a breakpoint P in the source code where execution is to stop." *Olsen*, 3:2-5.

"Assume the user sets a breakpoint at source line 2." *Olsen*, 4:30.

"When the user sets a breakpoint at source location P... *Olsen*, 4:64.

"Thus., when the user sets a source breakpoint at P" *Olsen* 5:27-30.

"Initially, the user sets a breakpoint P in the application source." *Olsen*, 12:1-2.

"the user enters a command "set breakpoint at P (step 50)." *Olsen*, 12:18-19, Fig. 4A, step 50.

PATENT
Atty. Dkt. No. ROC920010085US1
MPS Ref. No.: IBMK10085

None of these examples from *Olsen*, however, discloses a method for selecting a *particular point* within a program "P" to *actually set a breakpoint*. Applicants, however, claim a computer implemented method for selecting which branch points in the program being debugged to insert breakpoints. The actual branch points selected are determined by the subsequent steps of the method. Specifically, the branch points at which to insert a breakpoint are selected by (i) identifying a statement for the program being debugged, (ii) determining which basic block contains the statement, (iii) determining which other blocks present in the program control execution of the basic block, and (iv) inserting a breakpoint at each branch point contained in the other blocks controlling present in the program that control execution of the basic block.

The Examiner asserts that *Olsen* discloses the claimed limitation of "determining which blocks control execution of the basic block (See 12:30-24 and 40-45 and 7:7-65 – 8:27.)" The first passage cited by the Examiner, however, is directed to a method for setting "high and low water marks" in the machine code. The high and low water marks delimit regions of machine code that correspond to instructions that occur before or after the breakpoint, relative to the source code ordering of the instructions. The material cited by the Examiner includes a description of a process for identifying a high watermark that provides:

Then, debugger 30 determines the basic block containing commitpoint C and designates it "Cb". Further, a list of predecessor basic blocks, in predecessor first order, is determined and the list is designated as "W". The first basic block in W is designated as "p".

Next, debugger 30 retrieves the first machine code instruction in p, designating it as "i", and then retrieves the source statement (Si) which corresponds to machine code instruction i (step 104). Source statement Si and the corresponding machine code instruction i are retrieved from a table established by compiler 26 during the compile action.

Debugger 30 next determines whether a path exists from the machine code's entry point to machine code instruction "i" which does not contain a HW (decision step 108). If no (which means every path from the machine entry point to machine code "i" contains a HW), it proceeds to the next basic block (step 118) and continues at step 104.

Olsen, 12:24-35. The method begins by identifying a basic block *for an existing breakpoint*, and then proceeds to identify *all* of the predecessor blocks including instructions that occur prior to the commitpoint, relative to the source code order. The

Page 8

358823_1

PATENT
Att. Dkt. No. ROC920010085US1
MPS Ref. No.: IBMK10085

Examiner appears to assert that "determining which other blocks present in the program control execution of the basic block" equates to determining *all* paths of execution from the beginning of a procedure or program to the basic block that includes the identified statement. Applicants respectfully submit that this it is not the same as the claimed limitation of determining which other blocks in a program control execution of the basic block. Many blocks may be on a path that leads to the execution of the basic block, but not all of these paths control the execution of the basic block. This is the purpose of the Control Dependence Graph (CDG). Figures 2 and 3 of Applicants' specification illustrate the difference between paths to a block and control dependence of a block. According to the techniques disclosed by *Olsen*, the blocks on a path to block 7 include all blocks (1-6). That is, all blocks in predecessor order. However, as shown in the CDG in Figure 3 of Applicants' specification, block 7 is not control dependent on any of the other blocks; no matter what happens during program execution, block 7 will eventually be executed. Accordingly, no breakpoints are inserted in this block using the techniques of Applicants' invention. Using the techniques of *Olsen*, in view of *Shagam*, as a method for finding blocks on a path to block 7, however, a trace point would be placed in blocks 1-6. With Applicant's claimed method, no additional breakpoints would be needed at all, as blocks 1-6 have nothing to do with deciding whether block 7 is eventually executed or not.

Applicants respectfully submit, therefore, that creating a mapping (i.e., the high and low watermark region) to preserve the source order of instructions in optimized machine code fails to disclose the claimed limitation of determining which blocks control the execution of a basic block.

Further, the second passage cited by the Examiner is directed to emulating a portion of machine code instructions that have been moved to a location subsequent to a breakpoint, relative to a source-ordering of the instructions. *Olsen* discloses emulating these instructions to determine which path to take at a branch point during the execution of the program being debugged. Specifically, *Olsen* discloses that to handle this situation during debugging: "simply emulate all of the machine code instructions that affect the outcome of the branch in order to determine control flow and then throw away the emulated state." *Olsen*, 8:1-5. Respectfully, emulating machine

Page 9

358823_1

PATENT
Atty. Dkt. No. ROC920010085US1
MPS Ref. No.: IBMK10085

code instructions to determine what branch path to follow (i.e., to determine control flow during execution) fails to disclose the claimed limitation of determining which blocks control the execution of a basic block. Specifically, the former (control flow determinations) occurs during the execution of a program. For example, control flow determinations may be done by a debugger emulating instructions (that occur before the breakpoint in source-code order, but after the breakpoint in machine code order) to calculate values used to determine a branch path during program debugging. Whereas the latter, as claimed by Applicants, determines which blocks control the execution of the basic block based on the inherent structure of the program, as represented by the control dependence graph.

Finally, the Examiner concedes that:

Although *Olsen* discloses inserting commit points for all possible breakpoint locations (see, for example, column 8, lines 37-42) and determining exit points for the basic blocks (see, for example, column 10, lines 17-25), *Olsen* does not expressly disclose:

(d) inserting a breakpoint at each branch contained in the blocks controlling execution of the basic block.

The Examiner asserts, however, that *Shagam* "discloses inserting trace points at each block of execution" citing *Shagam*, 6:23-28, and that "trace points are analogous to breakpoints" citing *Shagam*, 2:9-31

Shagam places trace points in every block. In contrast, Applicants' invention places break points only in a small subset of the blocks along paths leading to the basic block. That subset is determined by identifying which blocks along those paths control eventual execution or non-execution of the basic block. Just being present along the path is not sufficient cause to insert a breakpoint, and would be extremely inefficient.

For all of the foregoing reasons, Applicants respectfully submit that claims 1, 12, and 16 are patentable over *Olsen* in view of *Shagam*. Applicants request, therefore, that the rejection of these claims be withdrawn.

Regarding claims 4 and 19, the Examiner asserts that *Olsen* discloses the additional limitation: wherein identifying the statement comprises identifying a statement

PATENT
Atty. Dkt. No. ROC920010085US1
MPS Ref. No.: IBMK10085

that may modify a program variable. To support this, the Examiner cites to *Olsen* 7:1-11. This material provides:

When the location of a variable X is to be found, the control flow graph is scanned backwards to find all the definition points for X that aren't superseded by following definition points for X. If all the definition points map X to the same location, then the value of X may be reconstructed. The control flow graph is then scanned forwards to see if this location for X is overwritten by a definition point for another variable, Y. If so, then X is dead, else its value is reported. As a special case, when the definition point for Y is emulated (instead of being executed), the change history can be used to ignore the emulation of this definition point.

Olsen, 7:1-11. Contrary to the Examiner's assertion that this material demonstrates that *Olsen* discloses the limitation "wherein identifying the statement comprises identifying a statement that may modify a program variable," this material is directed to finding the location of a variable during the execution of a program, (e.g., a memory address or a register value). Nothing in this passage discloses identifying statements in the source code of the program being debugged that may modify a selected variable; rather the cited material discloses how to determine a value to select a branch path during program execution. Accordingly, Applicants submit that claims 4 and 19 are patentable over *Olsen*, in view of *Shagam*, and request that the rejection be withdrawn.

Regarding claims 5 and 20, the Examiner asserts that *Olsen* discloses the additional limitation: wherein identifying the statement may modify a program variable comprises accessing a table composing the variable mapped to statements that may modify the variable. To support this assertion, the Examiner cites to *Olsen* 6:55-67. This cited material, however, is directed to using a change history mechanism used to recover the value of a variable at a breakpoint. *Olsen*, 6:55-63. Specifically *Olsen* discloses maintaining a history of machine instructions that are actually emulated and that change a variable value (or the state of a register): "a change record stores the modified values of registers and memory. This enables a roll back to a state as needed, when emulating [machine code] instructions that are out of order." *Olsen*, 6:51-55. Applicants, however, claim accessing a table comprising the variable mapped to source code statements that may modify the variable. Accordingly, Applicants submit that

PATENT
Atty. Dkt. No. ROC920010085US1
MPS Ref. No.: IBMK10085

claims 5 and 20 are patentable over *Olsen*, in view of *Shagam*, and request that the rejection be withdrawn.

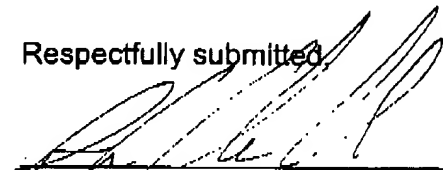
Regarding claims 2, 3, 6-11, 13-15, 17, 18 and 21-26, as each of these claims depends from one of independent claims 1, 12 or 16, Applicants submit that because *Olsen*, in view of *Shagam* fails to anticipate Applicants' invention, as claimed, the rejection of claims 2, 3, 6-11, 13-15, 17, 18 and 21-26 is obviated without the need for further remarks. Withdrawal of the rejection is respectfully requested.

Therefore, Applicants submit that claims 1-26 are patentable over *Olsen*, in view of *Shagam*. Withdrawal of the rejection is respectfully requested.

Conclusion

Having addressed all issues set out in the office action, Applicants respectfully submit that the claims are in condition for allowance and respectfully request that the claims be allowed.

Respectfully submitted,



Gero G. McClellan
Registration No. 44,227
MOSER, PATTERSON & SHERIDAN, L.L.P.
3040 Post Oak Blvd. Suite 1500
Houston, TX 77056
Telephone: (713) 623-4844
Facsimile: (713) 623-4846
Attorney for Applicants